Combining software agents and grid middleware

Richard Olejnik, Bernard Toursel[1], Maria Ganzha, and Marcin Paprzycki[2]

[1] Laboratoire d'Informatique Fondamentale, de Lille (LIFL UMR CNRS 8022)
Universite des Sciences et Technologies de Lille, USTL - Lille, France
{olejnik, toursel}@lifl.fr
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
{maria.ganzha, marcin.paprzycki}@ibspan.waw.pl

Abstract. Recently, the *Desktop-Grid ADaptive Application in Java* (*DG-ADAJ*) project has been unveiled. Its goal is to provide an environment which facilitates adaptive control of distributed applications written in Java for the Grid or the Desktop Grid. However, in its current state it can be used only in closed environments (e.g. within a single laboratory), as it lacks features that would make it ready for an "open Grid." The aim of this paper is to show how the *DG-ADAJ* can be augmented by usage of software agents and ontologies to make it more robust.

1 Introduction

The starting point for this research was development of Grid-enabled data mining software suite taking place within the *Distributed Data Mining* (*DisDaMin*) project (for details see [4, 5]). In conjunction, the *Desktop-Grid Adaptive Application in Java* (*DG-ADAJ*) project develops middleware platform for the Grid that, among others, could be used as a base for deployment of *DisDaMin* algorithms. It is the *DG-ADAJ* middleware that is of our particular interest in this paper. Specifically, we discuss how some of its natural shortcomings can be overcome by adding software agents as resource brokers and high level managers.

To achieve this goal we, first, present the *DG-ADAJ* project and discuss its most important features. We follow with a discussion of its shortcomings within an "open Grid." In the next section we describe an agent team based broker system and show how the two can be combined to create a robust Grid middleware.

2 *DG-ADAJ* Platform

Desktop Grid – Adaptive Distributed Application in Java (*DG-ADAJ*) is a middleware platform for Grid computing. It aims at facilitating a Single System Image (SSI) and enabling efficient execution of heterogeneous applications with irregular and unpredictable execution control. In Figure 1 we present the general overview of the *DG-ADAJ* architecture.

DG-ADAJ is an execution environment that is designed and implemented above the JavaParty and Java/RMI platforms according to a multi-layer structure, using several APIs (see Figures 1 and 2). One of its important features

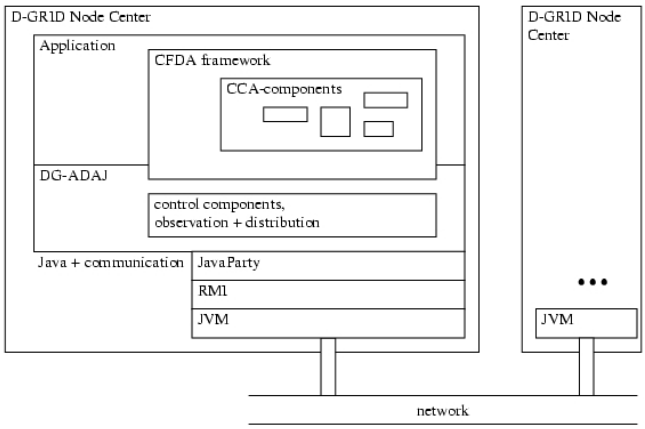

Fig. 1. DG-ADAJ Architecture.

are mechanisms based on control components (for more details of the *Common Component Architecture* (CCA), see [1]) for controlling granularity of computations and distribution of applications on the Desktop Grid platform. Note that use of components allows *DG-ADAJ* to be an environment for Java applications.

In addition to standard components, Super-Components have been developed to allow assembling together several components (they become *inner components* of a Super-Component). Super-Components implement framework services to manage their inner components. Specifically, connections between inner-components are achieved the same way as connection between standard components, while connections between inner-components and outer-components (components outside of the Super-Component) are achieved through a special mechanism of delegation between inner and outer ports (see Figure 3). Finally, the remote component is a special type of Super-Component which is implemented using the JavaParty notion of Remote class (defined using the JavaParty keyword *remote*).

DG-ADAJ runtime optimizes dynamic and static placement of the application objects within Java Virtual Machines of the Desktop Grid or the Grid [7]. Furthermore, *DG-ADAJ* provides special mechanisms, at the middleware level, which assure dynamic and automatic adaptation to variations of computation methods and execution platforms. This dynamic, on–line load balancing is based on object monitoring and relation graph optimization algorithms. Specifically, application observation mechanism in *DG-ADAJ* provides knowledge of behavior of the application during its execution. This knowledge is obtained by observation of object activity. A *DG-ADAJ* application comprises two types of objects: global and local. Global objects are observable, remote access and migratable. Local objects are traditional Java objects which are linked to a global object. Observation of a global object corresponds to monitoring its communication with other objects (global or local). Specifically, three components are

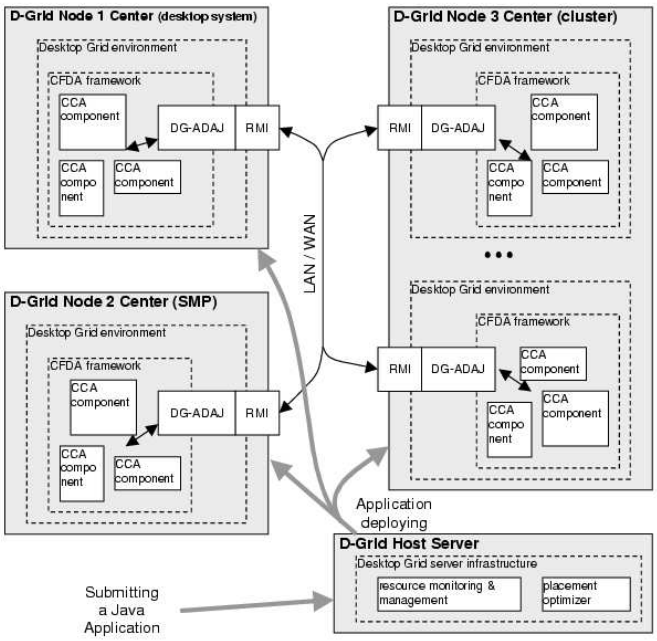

Fig. 2. The layered structure of the DG-ADAJ Environment.

used for the observation mechanism: (1) the *object graph*, which is built using relations between application objects, (2) the *relation tracer*, which stores information concerning these relations, and (3) the *observer*, which is responsible for the observation information update [8]). Observation of relationships between objects allows also computation of object activity (local and remote) representing their load. Overall, based on observations of object activity and on their relations, objects can be selected and moved from or to a computing node.

These mechanisms were experimented with in an earlier, built for cluster computing, version of *DG-ADAJ* (see, [6]). In the new version of *DG-ADAJ* load balancing takes into account also local load of each node, allowing computing nodes to be shared between several applications.

3 Agent brokers augmenting *DG-ADAJ*

Let us now assume that a *DisDamin* application is going to utilize the *DJ-ADAJ* environment to run within an "open Grid;" understood as a computational infrastructure consisting of nodes spread across the Internet. These nodes have different owners (including individuals who offer their home PC) that offer services and expect to be remunerated for their usage. In this case the Grid is a highly dynamic structure. There are two levels of dynamicity that can be observed. First, a given node suddenly becomes overloaded — when its owner

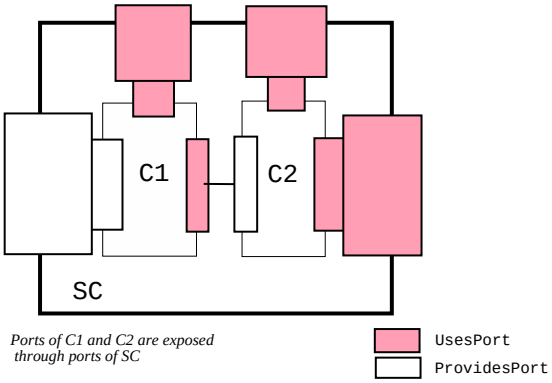

Fig. 3. *Super-component*

starts using it. Second, a given node disappears without a trace when the PC goes down due to a current spike. Interestingly, while the *DG-ADAJ* monitors performance of individual nodes and can deal with the first scenario, currently it cannot deal naturally with disappearing nodes. Observe that this is not a big problem in the case of a "closed Grid" e.g. in a laboratory, where all nodes are under some form of control of a system administrator.

Furthermore, *DG-ADAJ* does not include methods for resource brokering (which includes both resource description and matchmaking). While in a laboratory it is possible to know in advance, which machines will constitute the Grid, this is no the case in the "open Grid." Here, before any computational job is executed, nodes which will run it have to be found / selected first.

Finally, let us stress that resource brokering should involve an economic model, where resource providers are paid for rendered services. In return, quality of service (QOS) assurances have to be provided in a form of a service level agreement (SLA) "singed" by service-users and service-providers. These features are currently out of scope of the *DG-ADAJ* project.

In response to these "shortcomings" we propose to augment the *DG-ADAJ* with software agent "components." We follow here the proposal described in [2, 3], where more details of the agent-broker system can be found. Let us start with the use case diagram and a brief discussion of functionalities depicted there.

The main idea of the proposed system is utilization of agent teams consisting of a number of *worker agents* and a leader, the *LMaster agent*. It is the *LMaster* with whom *user agents* negotiate terms of task execution, and who decides whether to accept a new *worker agent* to the team. The *LMaster* agent has its mirror (*LMirror* agent). Its role is to be able to immediately take over — become the new *LMaster* — if the original *LMaster* goes down. In the case of *LMirror*'s disappearance, the *LMaster* immediately promotes one of *worker agents* to the role of *LMirror*. Note that an agent team may assure an SLA, as in the case when one machine/worker goes down, the *LMaster* is able to recognize the situation and redirect the job to another machine (and complete it almost on time).

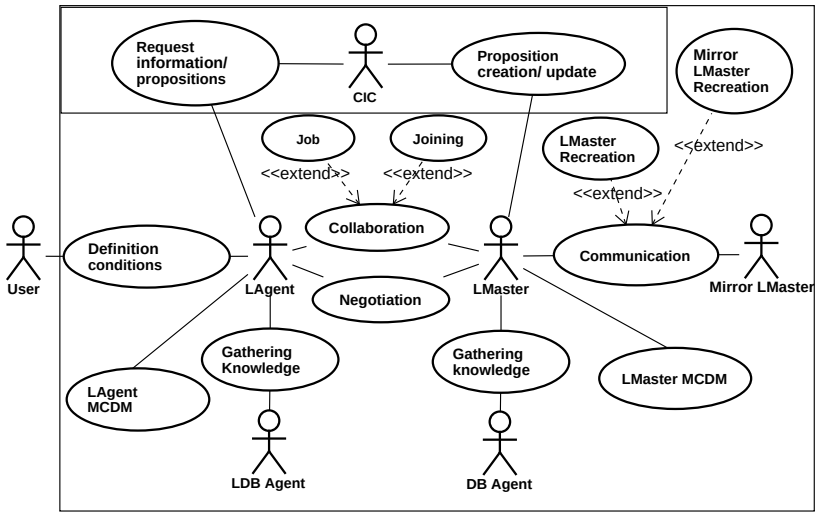

Fig. 4. Use Case diagram of the proposed system

For a team to be visible to potential users or team members, it must "post" its *team advertisement* for others to see. In our system (following results presented in [9]) we utilize a *yellow page* type approach and *LMaster* agents post their team advertisements within the *Client Information Center* (*CIC*). Such an advertisement contains information about offered resources (e.g. hardware capabilities, available software, price etc.) and / or "team metadata" (e.g. terms of joining, provisioning, specialization etc.). In this way *yellow pages* may be used: (1) by *user agents* looking for resources satisfying requirements of their task, and (2) by *worker agents* searching for a team to join. For example, *worker agent* representing computational resource with installed *DisDamin* software, may want to join a team specializing in solving problems utilizing *DisDamin* software.

Let us observe that in the case of a "closed Grid," this agent structure can be unchanged, though it also could be simplified. Here, instead of an evolutionary formation of agent teams (where workers and managers pick teams/agents of their linking), a team can be predefined by the administrator of the system. In this case also the *LMaster* and the *LMirror* agents can be selected to run on most stable (though not necessarily most powerful) machines. Overall, regardless of the scenario, the proposed approach adds a level of fault tolerance to the system and allows it to utilize Service Level Agreements and economic basis of functioning.

In the system, user initiates the execution of the job by providing its *user agent* with specific requirements such as: *resource requirements*—specification of resources needed to execute the task, and *execution constraints*—time, budget etc. From there on, the *user agent* acts autonomously. First, it queries the *CIC*

for resources matching requirements and obtains a list of query-matching teams. Then it negotiates with *LMaster*s representing selected teams, taking into account specified *execution constraints* to find the best team for the job. In the case of a closed environment it is possible to enforce that the (only existing/predefined by the administrator) agent team will execute the job.

Similarly, user can request that its agent joins a team, and specify conditions for joining (e.g. frequency of guaranteed jobs or share of generated revenue). In this case the *user agent* queries the *CIC* and obtains list of teams of interest; negotiates with them, decides which team to join and starts working for it. As stated above, in the case of a closed environment, the agent team(s) can be predefined. Observe that in both cases the economic model is taken into consideration.

To describe Grid resources we have decided to utilize ontologies. Unfortunately, there is no all-agreed ontology of the Grid and therefore we utilize an extremely simplified, RDF based, one [2]. What follows is an instance of that ontology describing worker *PC1541*, which has 16 Intel processors running at 3.0 GHz, 1 Gbyte of memory per processor, and 5 Gbytes of disk space available as a "Grid service:"

```
:LMaster3
  :hasContactAID
  ''monster@e-plant:1099/JADE'';
  :hasWorker  :PC1541.

:PC2929
  :a  :Computer;
  :hasCPU
  [
   a  :CPU;
   :hasCPUType  :Intel;
   :hasCPUFrequency  "3.0";
   :hasCPUnumber  "16";
 ]  ;
  :hasUserDiskQuota  "5000";
  :hasMemory  "1024".
```

Note that this simplistic ontology can be relatively easily replaced by a more realistic one as soon as such (all agreed by the Grid community) ontology becomes available. However, for the application like the *DisDamin* this ontology is quite sufficient as it specifies all the information necessary to perform initial distribution of data into computing nodes.

4 Combining agent-brokers and DG-ADAJ

Since agent-brokers and the DG-ADAJ are implemented in Java (recall that *DG-ADAJ* has been designed to facilitate programming of Java applications), combining them should be relatively easy. This is especially so that we have clearly delineated responsibilities. Agent-brokers act as "top level management"

and are responsible for resource brokering, setting the job to be executed and monitoring its successful completion. Components of *DG-ADAJ* are responsible for actually running the job. More specifically, in Figure 5 we depict how JADE agent platform ([10]) can be incorporated into the *DG-ADAJ* environment. Specifically, we propose that both the *DG-ADAJ* and JADE share the Java Virtual Machine and the RMI. In this way the RMI becomes the communication mechanism between the two environments.

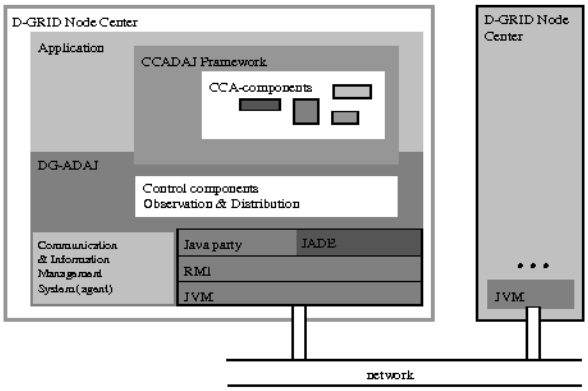

Fig. 5. *Introducing JADE agents into DG-ADAJ*

Taking this into account, we envision the following scenario taking place (in an open Grid system). User specifies the requirements for the data mining task. The *LAgent*s communicates with the *CIC* and obtains list of agent teams that are capable of executing this job. Then—using contract net protocol—the *LAgent* negotiates conditions of job execution (including the SLA) and picks one of them. Obviously, we assume that the selected team will run *DG-ADAJ* and the required application software. Information about the job is then transferred to the selected team. This information includes, among others, information where data sources are located. The *LMaster* communicates with selected *LAgent*s in its team (utilizing information about available machines—including information about workload obtained from the workload monitoring component of the *DG-ADAJ*), and decides which machines will be used to execute the job. Job information is send to *DG-ADAJ* components on selected machines and the job is left with them to execute. Upon completion of the job/task, the *DG-ADAJ* communicates with the *LAgent*s involved in the process. These agents confirm to the *LMaster* that the process is complete (and send to it the final result-set). The *LMaster*, in turn, communicates with the *LAgent* representing the user and completes all processes involved in finalizing the task (e.g. payment, results transfer etc.).

5 Concluding remarks

In this paper we have presented the *DG-ADAJ* project that provides middleware platform for the Desktop Grid and Grid. Our analysis indicated that, due to its underlying assumptions, the current state of the *DG-ADAJ* is lacking certain features to make it robust enough for the "open Grid." We have proposed to augment the *DG-ADAJ* with agent-brokers that will take care of high-level management functions, and with Grid resource ontology. We have also discussed how the two can be joined in a unified system. We are currently studying the specific way in which agent brokers can be implemented into the *DG-ADAJ* system and will report our progress in subsequent publications.

References

1. I.Alshabani, R. Olejnik and B. Toursel. Parallel Tools for a Distributed Component Framework *1st International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA04)*. Damascus, Syria, April 2004.
2. M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, M. Paprzycki, Utilizing agent teams in grid resource management — preliminary considerations, *Proceedings of the J. V. Atanasov Conference*, IEEE CS Press, Los Alamitos, CA, 2006, 46-51
3. M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, M. Paprzycki, Efficient Matchmaking in an Agent-based Grid Resource Brokering System, *Proceedings of the International Multiconference on Computer Science and Information Technology*, PTI Press, 2006, 327-335
4. V. Fiolet and B. Toursel, *Distributed Data Mining*, In Scalable Computing: Practice and Experiences, Vol. 6, Number 1, March 2005, pp. 99-109.
5. V. Fiolet and B. Toursel, *Progressive Clustering for Database Distribution on a Grid*, In Proc. of ISPDC 2005, IEEE Computer Society, july 2005, pp. 282-289.
6. R. Olejnik, A. Bouchi, B. Toursel. Object observation for a java adaptative distributed application platform. *Intl. Conference on Parallel Computing in Electrical Engineering PARELEC 2002*, pp. 171-176., Warsaw, Poland, September 2002.
7. E. Laskowski, M. Tudruj, R. Olejnik, B. Toursel. *Bytecode Scheduling of Java Programs with Branches for Desktop Grid. to appear in the Future Generation Computer Systems*, Springer Verlag.
8. A. Bouchi, R. Olejnik and B.Toursel. *A new estimation method for distributed Java object activity.* 16th International Parallel and Distributed Processing Symposium, Marriott Marina, Fort Lauderdale, Florida, April 2002.
9. Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In: *Proceedings of the WWW'02: International World Wide Web Conference*, Hawaii, USA. ACM Press, New York, USA, pp.89-98, 2002.
10. JADE: Java Agent Development Framework. See `http://jade.cselt.it`